# SAS® Viya® 3.5 Administration:
## CAS Authorization

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CAS Authorization: Overview</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>CAS Authorization: How To (Authorization Window)</strong></td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Navigation</td>
<td>3</td>
</tr>
<tr>
<td>Examine Access</td>
<td>3</td>
</tr>
<tr>
<td>Set an Access Level</td>
<td>4</td>
</tr>
<tr>
<td>Add a Direct Access Control</td>
<td>5</td>
</tr>
<tr>
<td>Remove a Direct Access Control</td>
<td>6</td>
</tr>
<tr>
<td>Remove Multiple Direct Access Controls</td>
<td>6</td>
</tr>
<tr>
<td>Examples: Manage Access to a Caslib</td>
<td>6</td>
</tr>
<tr>
<td>Provide Row-Level (Filtered) Access</td>
<td>8</td>
</tr>
<tr>
<td>Identify the Source of Effective Access</td>
<td>9</td>
</tr>
<tr>
<td><strong>CAS Authorization: How To (CLI)</strong></td>
<td>9</td>
</tr>
<tr>
<td>Getting Access Information</td>
<td>9</td>
</tr>
<tr>
<td>Managing Access Controls</td>
<td>10</td>
</tr>
<tr>
<td>Managing Access Controls (Bulk Approach)</td>
<td>11</td>
</tr>
<tr>
<td>Details and Tips</td>
<td>12</td>
</tr>
<tr>
<td><strong>CAS Authorization: Concepts</strong></td>
<td>15</td>
</tr>
<tr>
<td>Scope</td>
<td>15</td>
</tr>
<tr>
<td>Key Terms</td>
<td>15</td>
</tr>
<tr>
<td>Principals</td>
<td>16</td>
</tr>
<tr>
<td>Administrators</td>
<td>16</td>
</tr>
<tr>
<td>Inheritance</td>
<td>16</td>
</tr>
<tr>
<td>Permissions</td>
<td>17</td>
</tr>
<tr>
<td>Permissions by Task</td>
<td>18</td>
</tr>
<tr>
<td>Row-Level Access</td>
<td>20</td>
</tr>
<tr>
<td>Column-Level Access</td>
<td>24</td>
</tr>
<tr>
<td>Access to Actions</td>
<td>25</td>
</tr>
<tr>
<td>Authorization Decisions</td>
<td>26</td>
</tr>
<tr>
<td>Access Control Transactions</td>
<td>27</td>
</tr>
<tr>
<td>Origins of Effective Access</td>
<td>27</td>
</tr>
<tr>
<td>Reduced Visibility: Hidden Caslibs</td>
<td>28</td>
</tr>
<tr>
<td>Application and Persistence of CAS Access Controls</td>
<td>29</td>
</tr>
</tbody>
</table>
CAS Authorization: Overview

To learn about the Cloud Analytic Services (CAS) authorization system, see Concepts.

To manage access, use the interface that best meets your needs. Here are suggestions:

- To interactively manage access, use the Authorization window.
- To script management of CAS access controls in a full deployment, use the command-line interface.
- For comprehensive programmatic management of CAS access controls, use CAS actions. See the Access Control action set in SAS Viya: System Programming Guide.

**CAUTION**

Do not rely exclusively on CAS access controls to protect data. See “CAS Authorization: Host Access”.

CAS Authorization: How To (Authorization Window)

Introduction

These instructions explain how to manage access to caslibs, tables, and rows using the Authorization window.
Navigation

Here is one way to access the Authorization window for caslibs and tables:

1. In the applications menu (≡), under ADMINISTRATION, select Manage Environment.
2. In the vertical navigation bar in SAS Environment Manager, click .
3. On the Data page, locate and select a global caslib or table.
4. Right-click, and select View authorization or Edit authorization.

Note: If Edit authorization is not available, you are not authorized to modify access to the selected object.

Examine Access

Scope

The scope of the display is as follows:

- There is always a row for Authenticated Users.
- There is always a row for you, the currently connected user who is using the display.
- There is a row for each principal that is assigned to an access control that affects access to the current object.
- If you add an identity and do not give that identity at least one direct setting, that identity is automatically removed from the display.
- You cannot directly remove a row. If you remove all direct settings for an identity and there is no other reason for that identity to be displayed, that identity is automatically removed from the display.
- Only the permissions that are relevant for an object (directly or for inheritance purposes) are displayed for that object.
- The display does not reflect the impact of CAS role membership or status.

Permissions

For each principal and permission, the following icons describe effective (net) access to the current caslib or table.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>Authorized</td>
</tr>
<tr>
<td>Icon</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>---------------------</td>
</tr>
<tr>
<td>❌</td>
<td>Not Authorized</td>
</tr>
<tr>
<td>🔍</td>
<td>Row-Level</td>
</tr>
<tr>
<td>🤷</td>
<td>Unknown</td>
</tr>
<tr>
<td>🔶</td>
<td>Direct(^1)</td>
</tr>
</tbody>
</table>

\(^1\) Indicates that a permission is directly assigned to the specified principal on the current object.

**Access Levels**

The **Access Level** column provides an alternative to interacting with individual permissions.  
- When you manage access, each access level is a shortcut for adding a set of direct access controls.  
- When you view access information, each access level is a shorthand description of a set of effective permissions.

**Table 1  Access Levels**

<table>
<thead>
<tr>
<th>Access Level</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>No access</td>
<td>None</td>
</tr>
<tr>
<td>Read</td>
<td>Only ReadInfo and Select</td>
</tr>
<tr>
<td>Write</td>
<td>All except ManageAccess and AlterCaslib</td>
</tr>
<tr>
<td>Full control</td>
<td>All</td>
</tr>
<tr>
<td>Custom</td>
<td>Any other combination, including any row-level access</td>
</tr>
</tbody>
</table>

**Note:** Access levels exist only in the presentation layer in SAS Environment Manager. CAS stores and evaluates individual permissions, not cumulative access levels.

**Set an Access Level**

1. Open the Edit Authorization window for a CAS object.
2. If the principal that you want to work with is not already listed, click Add. In the Add Identities window, move the user or group to the right pane, and click OK.
Note: If guest access is enabled, you must select Add Identities after you click 🌐. Or, if you need to add Guest to the display, select Add Guest after you click 🌐.

3 In the Access Level column, click and drag a gauge to adjust access.

Note: If a gauge is not displayed, select an access level other than (custom) from the drop-down list.

Note: Each time you change an access level, direct access controls are added as needed to meet the definition of the new access level. If you want to discard all unsaved changes, click Cancel.

CAUTION
Reducing the access level for a group that you belong to might block your access. To preserve your access, make sure you have a higher precedence (offsetting) direct grant. If you are a Superuser, this precaution is not strictly necessary.

4 If you modified access for a group, click Preview. Examine the impact of the change on other principals. For example, increasing the access level for Authenticated Users from No access to Full control affects all authenticated users who do not have a more specific denial.

5 Click Save.

Add a Direct Access Control

1 Open the Edit Authorization window for a CAS object.

2 If individual permissions are not already displayed, select the Show individual permissions check box.

3 If the principal that you want to work with is not already listed, click 🌐. In the Add Identities window, move the user or group to the right pane, and click OK.

Note: If guest access is enabled, you must select Add Identities after you click 🌐. Or, if you need to add Guest to the display, select Add Guest after you click 🌐.

4 Click the effective access icon (for example, ☑️) for the principal and permission that you want to modify.

5 In the pop-up window, select Grant or Deny in the Direct Setting drop-down list.

CAUTION
Before you deny access for a group that you belong to, make sure you have a higher precedence (offsetting) direct grant. If you are a Superuser, this precaution is not strictly necessary.

6 If you modified access for a group, click Preview in the Edit Authorization window.
Notice that a diamond is displayed in the cell that you modified. The diamond indicates that effective access comes from a direct setting.

Examine the impact on other principals. For example, a direct denial for GroupA affects all members of GroupA who do not have their own direct settings.

Click Save.

Remove a Direct Access Control

1. Open the Edit Authorization window for a CAS object.
2. In a cell that includes a diamond, click the effective access icon.
3. In the pop-up window, select (none) from the Direct settings drop-down list.
4. In the Edit Authorization window, notice that the new effective access value is unknown (ıld parasite). Click Preview.
   - Notice that the new effective access value is known. Or, if you removed the only setting that made the associated user or group a relevant principal for the current object, the user or group is no longer included in the display.
   - If you modified access for a group, examine the impact on other principals.
5. Click Save.

Remove Multiple Direct Access Controls

1. Open the Edit Authorization window for a CAS object.
2. In a row that includes at least one direct setting, click the first cell. The row is selected.
3. Click to remove all direct access controls for the selected identity.
4. Notice that effective access for any affected cells is unknown (ıld parasite). Click Preview.
   - Notice that all effective access values are known. Or, if the associated user or group is no longer a relevant principal for the current object, the user or group is no longer included in the display.
   - If you modified access for a group, examine the impact on other principals.
5. Click Save.

Examples: Manage Access to a Caslib

Provide Public Access to a Caslib

To give all users Read access to a new global caslib that you added:
1 Open the caslib’s Edit Authorization window. (See Navigation.)

2 In the row for Authenticated Users, increase the Access Level to Read.

3 Click Save.

Provide Selective Access to a Caslib
To give a particular user Read and Write access to a new global caslib that you added:

1 Open the caslib’s Edit Authorization window. (See Navigation.)

2 Click \( \text{add} \) in the table toolbar.

3 In the left pane of the Add Identities window, locate the user. Move the user to the right pane, and click OK.

4 In the Edit Authorization window, increase the user’s Access Level to Write.

5 Click Save.

Block All Access to a Caslib
To block all access for a particular identity:

1 Open the caslib’s Edit Authorization window. (See Navigation.)

2 If the identity is not already listed, click \( \text{add} \) in the table toolbar.

   In the left pane of the Add Identities window, locate the user, group, or custom group that you want to block. Move that identity to the right pane, and click OK.

3 In the Edit Authorization window, decrease the identity’s Access Level to None.

4 If the identity is not an individual user, click Preview. Examine the impact of your change on other listed identities.

5 Click Save.

Limit Write Access to a Caslib
To allow only Read access for a particular identity:

1 Open the caslib’s Edit Authorization window. (See Navigation.)

2 If the identity is not already listed, click \( \text{add} \) in the table toolbar.

   In the left pane of the Add Identities window, locate the user, group, or custom group that you want to block. Move that identity to the right pane, and click OK.

3 In the Edit Authorization window, decrease the identity’s Access Level to Read.

4 If the identity is not an individual user, click Preview. Examine the impact of your change on other listed identities.

5 Click Save.
Provide Row-Level (Filtered) Access

To make different subsets of rows available to different identities, set one or more row-level grants. Each row-level grant includes a filter that limits the available rows.

1. Open the Edit Authorization window for a CAS table.

2. If the principal that you want to work with is not already listed, click Add. In the Add Identities window, move the user or group to the right pane, and click OK.

   Note: If guest access is enabled, you must select Add Identities after you click Add. Or, if you need to add Guest to the display, select Add Guest after you click Add.

3. If individual permissions are not displayed, select the Show individual permissions check box.

4. In the Select column, click an effective access icon.

5. In the pop-up window, select Row-level Grant from the Direct setting drop-down list.

6. In the Row-Level Filter window:
   a. Specify an expression that includes only the rows that the principal should be able to access. The basic format is: column-name operator value. Here are basic examples:

<table>
<thead>
<tr>
<th>Type of Filter</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>sales&lt;1000</td>
</tr>
<tr>
<td>Character</td>
<td>Make='Ford'</td>
</tr>
<tr>
<td>Dynamic</td>
<td>user='SUB::SAS.Userid'</td>
</tr>
</tbody>
</table>

   For details, see Row-Level Access.

   Note: If you view or edit a filter that was initially created programmatically, you might see escape characters and a different pattern of quotation marks.

   b. Click OK.

7. In the Edit Authorization window, next to the new setting, notice that a diamond is displayed. The diamond indicates that effective access comes from a direct setting.

8. If you modified access for a group, click Preview. Examine the impact on other principals.

9. Click Save.
Identify the Source of Effective Access

To determine which access control causes a particular effective access result, examine the origins information for that result.

1. Open the View Authorization window for the target CAS object.
2. Click the effective access icon for which you want origins information.
3. In the pop-up window, next to the Effective Access value, click  

Note: The icon is disabled if you have changes that you have neither saved nor previewed.

4. In the Origins window, review the displayed information.
   - The Source object field indicates where the determinative access control is set.
   - The Principals field indicates which identity the determinative access control are assigned to.

Note: If multiple access controls of equal precedence cause the result, multiple principals are listed.

For more information, see Origins of Effective Access.

CAS Authorization: How To (CLI)

The following examples assume that you have already signed in to SAS Viya at the command line. See the preliminary instructions in SAS Viya Administration: Using the Command-Line Interfaces

Getting Access Information

Example: List tableA’s direct access controls.

```
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableA.sashdat
```

Example: List tableA’s direct access controls and inherited settings.

```
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableA.sashdat --list-type all
```

Example: Show effective (net) access to tableA for userA.

```
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableA.sashdat --control-type effective --user userA
```

Example: Show effective access to tableA for groupA and groupB.
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableA.sashdat --control-type effective --group 'groupA|groupB'

**Example:** Show the source of userA's access to tableA.
```
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableA.sashdat --control-type origin --user userA
```

### Managing Access Controls

**Example:** Remove all direct access controls from tableA.
```
sas-admin cas tables clear-controls --server serverA --caslib caslibA --table tableA.sashdat
```

**Example:** Set a simple row-level access control on the CARS table so that members of groupA can see only those rows where the value in the Make column is *Ford*.
```
sas-admin cas tables add-control --server serverA --caslib caslibA --table CARS.sashdat --group groupA --grant Select --where "make='Ford'"
```

**Example:** Set an identity-based, row-level access control on the Salary table. The reason is so that each authenticated user can see only those rows where the value in the User column is their own user ID.
```
sas-admin cas tables add-control --server serverA --caslib caslibA --table salary.sashdat --group "*" --grant Select --where "User='SUB::SAS.Userid'"
```

**Example:** Enable guests to read data in the Public caslib.

1. `sas-admin cas caslibs add-control --server serverA --caslib Public --guest --grant ReadInfo --superuser`
2. `sas-admin cas caslibs add-control --server serverA --caslib Public --guest --grant Select --superuser`

This example is applicable to only a deployment where guest access is enabled.

In the standard configuration, because only a privileged user can modify access to the Public caslib, the superuser option is specified here. Only a member of the Superuser role for the specified CAS server can obtain elevated privileges by specifying the superuser option.

The grants in this example support reading of data, but do not support just-in-time loading of data. Instead of granting LimitedPromote to guest, consider using a different technique for ensuring that data is loaded.

Because this example does not create and reuse a dedicated superuser session, you must specify the superuser option in each command where elevated privileges are needed.

**Example:** Enable groupA to read data (and perform just-in-time data loading) in a new caslib.
```
sas-admin cas caslibs add-control --server serverA --caslib caslibA --group groupA --grant ReadInfo
sas-admin cas caslibs add-control --server serverA --caslib caslibA --group groupA --grant Select
sas-admin cas caslibs add-control --server serverA --caslib caslibA --group groupA
```

1 This example is applicable to only a deployment where guest access is enabled.

2 The grants in this example support reading of data, but do not support just-in-time loading of data. Instead of granting LimitedPromote to guest, consider using a different technique for ensuring that data is loaded.

Because this example does not create and reuse a dedicated superuser session, you must specify the superuser option in each command where elevated privileges are needed.
--grant LimitedPromote

**Example:** Make the same changes as in the preceding example, but use an access control transaction so that you can review your changes before you commit them to the server.

1. `sas-admin cas sessions create --name mysess --server serverA --superuser caslibA`
2. `sas-admin cas transactions checkout --session-id XYZ --server serverA --caslib caslibA`
3. `sas-admin cas caslibs add-control --session-id XYZ --server serverA --caslib caslibA --group groupA --grant ReadInfo`
4. `sas-admin cas caslibs add-control --session-id XYZ --server serverA --caslib caslibA --group groupA --grant Select`
5. `sas-admin cas caslibs add-control --session-id XYZ --server serverA --caslib caslibA --group groupA --grant LimitedPromote`
6. `sas-admin cas caslibs list-controls --session-id XYZ --server serverA --caslib caslibA`
7. `sas-admin cas transactions commit --session-id XYZ --server serverA`
8. `sas-admin cas sessions delete --session-id XYZ --server serverA`

1. Start a session. If you are a member of the Superuser role for the associated CAS server, give the session Superuser status.
2. Check out the caslib into the session that you just started. Use the session ID that is returned from the preceding command. The session-id value `XYZ` is used here for simplicity. Checking out an object automatically starts a transaction.
3. Grant access within the transaction.
4. Grant access within the transaction.
5. Grant access within the transaction.
6. Review the results. Because you supply the session ID, the output reflects the uncommitted changes in your session.
7. After you review the output from the list-controls command, commit your changes.
8. If you are finished, it is a good practice to delete your session.

---

**Managing Access Controls (Bulk Approach)**

You can use the bulk approach to replace, add, or remove multiple access controls from a specified table or caslib.

---

**Note:** The bulk approach requires a JSON file as input. The JSON file can be referenced with an absolute path or a path that is relative to the location of the CLI executable (sas-admin). See "**JSON File Format**".

---

**Example:** Replace any direct access controls on `tableA` with access controls from an external JSON file. In this example, the replacement access controls are derived from `tableB` and are then applied to `tableA`.

1. `sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableB.sashdat > ac.json`
2. `sas-admin cas tables replace-controls --server serverA --caslib caslibA --table tableA.sashdat --source-file ac.json`
This example writes the direct access controls for tableB to the file `ac.json` in the directory from which you are running your CLI.

Note: This example assumes that the profile that you are using specifies `json` as your default output type. Otherwise, you must use the global option `--output` to specify `json` as the output type for this command. That option must immediately follow `sas-admin`.

Note: You can modify the output from tableB before you use it to replace direct access controls on tableA.

Delete any direct access controls on tableA, and replace them with the access controls that you wrote to the `ac.json` file.

Review the new set of direct access controls on tableA.

**Example:** Delete multiple access controls, as specified in a referenced JSON file.

```bash
sas-admin cas tables list-controls --server serverA --caslib caslibA --table tableB.sashdat
> oldac.json
```

```bash
sas-admin cas tables remove-controls --server serverA --caslib caslibA --table tableA.sashdat
--source-file oldac.json
```

Write the direct access controls for tableA to a JSON file.

Note: This example assumes that the profile that you are using specifies `json` as your default output type. Otherwise, you must use the global option `--output` to specify `json` as the output type for this command. That option must immediately follow `sas-admin`.

Note: You can modify the JSON file before you use it to remove direct access controls from tableA. For example, if you want to retain some of the access controls, delete those entries from the JSON file.

Use the JSON file to delete access controls from tableA.

**Example:** Add multiple access controls, as specified in a referenced JSON file.

```bash
sas-admin cas tables add-controls --server serverA --caslib caslibA --table tableA.sashdat
--source-file newac.json
```

---

**Details and Tips**

**Basics**

- You can add, delete, and replace only direct access controls.
- You cannot modify access that a caslib inherits.
To modify access that a table inherits, set direct access controls on the parent caslib.

A request to delete a direct access control that does not exist does not generate an error.

Use of access control transactions is optional. You do not have to check out an object in order to modify its access controls.

In the list-controls command, use the control-type and list-type options as follows:
- Use the control-type option only if you want to obtain net access information (effective) or source information (origins).
- The list-type options are not relevant if the control-type is effective or origin.
- Use the list-type option only if you want to obtain inherited settings, in addition to direct access controls (all) or instead of direct access controls (inherited).

You can obtain origins on page 27 information for only one identity at a time.

In the examples, the value serverA is used for simplicity. A more typical server name is cas-shared-default.

In the examples that target tables in path-based caslibs, source files are referenced (for example, --table tableA.sashdat) instead of in-memory names (for example, --table tableA). This reflects the best practice that is described in “Application and Persistence of CAS Access Controls”.

Note: The examples use SASHDAT files. All source file types that CAS supports (such as CSV and SAS7BDAT) are supported in the CLI.

For information about using environment variables, see SAS Viya Administration: Using the Command-Line Interfaces.

Throughout this topic, the term access control refers to an access control in the CAS authorization system. To manage access to content objects and functionality, use the general authorization system.

Principals

To specify a particular identity (where supported), you must provide a user ID or a group ID rather than a name.

CAUTION
The user ID and the group ID that you provide are not validated. Make sure the IDs that you provide are accurate.

To specify multiple users or multiple groups (where supported), use the pipe character (|) as a delimiter and enclose the string in single quotation marks (for example: --user 'userA|userB'). You cannot specify both users and groups in a single request.

Note: Specifying multiple users or multiple groups is supported for only requests for effective access information.

The group * corresponds to Authenticated Users. To specify that principal, enter --group '***'.

The Guest principal represents all users who connect as guests. To specify that principal, enter the option --guest and do not specify a value.
Permissions

- To specify a particular permission (where supported), use one of the following case-insensitive values: ReadInfo, Select, LimitedPromote, Promote, CreateTable, DropTable, DeleteSource, Insert, Update, Delete, AlterTable, AlterCaslib, or ManageAccess. You cannot specify multiple permissions in a single request.
- For information about the scope and purpose of each permission, see “Permissions” on page 17.

JSON File Format

The bulk approach to managing access controls requires a JSON file as input. To create a sample JSON file, add direct access controls to any global caslib or table, and use the list-controls command with JSON output. See “Managing Access Controls (Bulk Approach)”. Here is an example:

```json
{
    "items": [
        {
            "identity": "groupA",
            "identityType": "group",
            "permission": "readInfo",
            "type": "grant",
            "version": 1
        },
        {
            "identity": "groupA",
            "identityType": "group",
            "permission": "select",
            "type": "grant",
            "version": 1
        },
        {
            "identity": "userA",
            "identityType": "user",
            "permission": "insert",
            "type": "grant",
            "version": 1
        },
        {
            "identity": "userA",
            "identityType": "user",
            "permission": "delete",
            "type": "grant",
            "version": 1
        }
    ]
}
```

Fine-Grained Controls

- Row-level grants are always for the Select permission on a table. The syntax for row-level permission filters is the same as in other CAS authorization interfaces.
You cannot set column-level permissions using this interface.

CAS Authorization: Concepts

Scope

CAS authorization manages access to the following CAS objects:
- caslibs
- CAS tables and columns
- CAS action sets and actions

CAS authorization requirements do not apply in the following circumstances:
- The requesting user has assumed a role that is exempt from all applicable authorization requirements. For example, the user has assumed the Superuser role and the request is to add a caslib.
- The target object is not potentially sharable. For example, the target is a table in a personal caslib, a session caslib, or the session scope of a global caslib.

Note: Not all interfaces expose all aspects of CAS authorization.

Key Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Access control | A composite of authorization elements.  
  Example: An access control grants ReadInfo to groupA on caslibA. |
| Target    | A resource.  
  Examples: tableA, caslibA |
| Principal | The user, group, or construct to which an access control is assigned.  
  Examples: UserA, GroupA, Authenticated Users |
| Permission | A type of access.  
  Values: ReadInfo, Select, LimitedPromote, Promote, CreateTable, DropTable,  
  DeleteSource, Insert, Update, Delete, AlterTable, AlterCaslib, ManageAccess |
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting</td>
<td>An indication of whether (and to what extent) access is provided.</td>
</tr>
<tr>
<td></td>
<td>Values: Grant, Row-Level Grant, Deny</td>
</tr>
<tr>
<td>Filter</td>
<td>In a row-level grant of the Select permission, the constraint expression.</td>
</tr>
<tr>
<td></td>
<td>Example: User='SUB::SAS.Userid', sales&gt;1000</td>
</tr>
<tr>
<td>Effective access</td>
<td>A context-neutral description of the net result of all relevant access controls.</td>
</tr>
<tr>
<td></td>
<td>Values: Authorized, Not Authorized, Row-Level</td>
</tr>
<tr>
<td>Access outcome</td>
<td>In an access request, the authorization decision.</td>
</tr>
<tr>
<td></td>
<td>Values: Authorized, Not Authorized, Row-Level Authorization</td>
</tr>
</tbody>
</table>

**Principals**

The principal in an access control is the user, group, or construct to which the access control is assigned. The CAS authorization system supports the following principals:

- an individual authenticated user
- a user group (a custom group or a group in your authentication provider)
- Authenticated Users (the construct that represents all authenticated users)

**Note:** In some programmatic contexts, this construct corresponds to the group that is named `*`.

- Guest (the identity type that facilitates guest access)

**Note:** Guest is not part of Authenticated Users.

**Administrators**

CAS roles provide per-server assumable access to administrative functionality. For example, the Superuser role is exempt from authorization requirements throughout a CAS server, except for data access requests. See [SAS Viya Administration: SAS Cloud Analytic Services](#).

**Inheritance**

Access flows through a hierarchy of objects. Each parent object conveys settings to its child objects. Each child object inherits settings from its parent object.

Here are the inheritance relationships:
- Access flows from a caslib to its tables.
- Access flows from a table to its columns.
- Access flows from an action set to its actions.

Note: Each caslib always has inherited denials of all permissions for Authenticated Users. Those inherited denials prevent access if there are no higher precedence grants.

---

**Permissions**

<table>
<thead>
<tr>
<th>Permission</th>
<th>Data Enforcement Levels</th>
<th>Affected Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Caslib</td>
<td>Table</td>
</tr>
<tr>
<td>ManageAccess</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ReadInfo</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LimitedPromote</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Promote</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CreateTable</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DeleteSource</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>DropTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>AlterCaslib</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>AlterTable</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Insert</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

1 You can set permissions at or above the level where they are enforced. See also [Access to Actions](#).
2 If no table is specified in the request, LimitedPromote is checked at the caslib level.
3 To delete any direct access controls, the ManageAccess permission is required.
Permissions by Task

To complete a task, you must have sufficient access to all relevant data objects. The following tables document the minimum set of permissions that are required for selected tasks.

**Table 2  Simple Tasks**

<table>
<thead>
<tr>
<th>Task (CAS Action)</th>
<th>Caslib</th>
<th>Table</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set caslib permissions</td>
<td>ReadInfo</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ManageAccess</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify caslib properties</td>
<td>ReadInfo</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>AlterCaslib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set table permissions</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ManageAccess</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify table properties¹</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AlterTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load a table from a caslib’s data source</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td>(loadTable)</td>
<td></td>
<td>Select</td>
<td></td>
</tr>
<tr>
<td>Transfer and load an entire file (upload)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td>Transfer rows to the server (addTable)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move a table to global scope (promote)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Promote²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove a table from global scope (dropTable)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DropTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete a file (deleteSource)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DeleteSource</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(ManageAccess)³</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persist a file (save)</td>
<td>ReadInfo</td>
<td>ReadInfo</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CreateTable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(DeleteSource)⁴</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3  Compound Tasks

<table>
<thead>
<tr>
<th>Task (CAS Actions)</th>
<th>Required Permissions</th>
</tr>
</thead>
</table>
| New import (upload + save + loadTable + promote) | Caslib: ReadInfo, CreateTable, LimitedPromote\(^1\)  
Table: ReadInfo, Select |
| Re-import (upload + save + dropTable + loadTable + promote) | Caslib: ReadInfo, CreateTable  
Table: ReadInfo, Select, DeleteSource, DropTable, LimitedPromote\(^2\) |
| Just-in-time load (loadTable + promote) | Caslib: ReadInfo  
Table: ReadInfo, Select, LimitedPromote\(^3\) |
| Delete a global-scope table (dropTable + deleteSource) | Caslib: ReadInfo  
Table: ReadInfo, DeleteSource, DropTable |

1 LimitedPromote is granted on the caslib because the table does not exist in the caslib at the time that the task is initiated.  
2 DeleteSource is required to replace the source table. DropTable is required to replace the global-scope table.  
3 LimitedPromote is sufficient for same-caslib promotion (but not for cross-caslib promotion).

**TIP** In an import that omits the save action, promotion requires the Promote permission. In such an import, the LimitedPromote permission is insufficient because promotion is not from a source file in the target caslib.
Row-Level Access

Overview of Row-Level Access

A row-level grant includes a filter that limits the Select permission on a table. A user who has row-level access to a table can view only those rows that are within the associated filter. See also Application and Persistence.

For example, you can use a row-level grant to enable groupA to see only those rows in tableA where the value in the Toy_Price column is 25. Here is an overview of the process:

1. On tableA, give groupA a row-level grant of Select permission.
   Specify the following filter: `Toy_Price=25`
   
   Note: For detailed instructions, see “Provide Row-Level (Filtered) Access”.

2. Make sure that groupA has ReadInfo access to tableA and its parent caslib.

3. Make sure that groupA is not a member of another group that has a grant or denial of the Select permission on tableA.

4. Verify that when a member of groupA accesses tableA, the expected rows are returned.

Syntax for Row-Level Filters

<table>
<thead>
<tr>
<th>Operator (Alias)</th>
<th>Example Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains (?)</td>
<td><code>Toy_Type Contains 'cars'</code></td>
</tr>
<tr>
<td>Not Contains</td>
<td></td>
</tr>
<tr>
<td>In</td>
<td><code>Toy_Type In ('dolls' 'cars' 'animals')</code></td>
</tr>
<tr>
<td>Not In</td>
<td></td>
</tr>
<tr>
<td>Between -inclusive</td>
<td><code>Toy_Price Between 20 AND 30</code></td>
</tr>
<tr>
<td>Not Between -inclusive</td>
<td></td>
</tr>
<tr>
<td>Like</td>
<td><code>Toy_Type Like 'd%'</code></td>
</tr>
<tr>
<td>Operator (Alias)</td>
<td>Example Filter</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>= (EQ)</td>
<td>Toy_Price=25</td>
</tr>
<tr>
<td>&gt; (GT)</td>
<td></td>
</tr>
<tr>
<td>&lt; (LT)</td>
<td></td>
</tr>
<tr>
<td>=&gt; (GE)</td>
<td></td>
</tr>
<tr>
<td>&lt;= (LE)</td>
<td></td>
</tr>
<tr>
<td>^= (NE, ~=)</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Profit &gt; (Sales * .5)</td>
</tr>
<tr>
<td>-</td>
<td>Toy_Type='cars' OR Toy_Type='dolls'</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>**</td>
<td></td>
</tr>
<tr>
<td>()</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>AND (&amp;)</td>
<td></td>
</tr>
<tr>
<td>OR (</td>
<td>, !)</td>
</tr>
<tr>
<td>NOT</td>
<td></td>
</tr>
<tr>
<td>Is Missing</td>
<td>Toy_Type Is Not Null</td>
</tr>
<tr>
<td>Is Not Missing</td>
<td></td>
</tr>
<tr>
<td>Is Null</td>
<td></td>
</tr>
<tr>
<td>Is Not Null</td>
<td></td>
</tr>
</tbody>
</table>

**Identity-Based Substitution**

Identity-based substitution is a powerful and concise technique for defining row-level access. You can use substitution to implement any number of per-user access distinctions with a single row-level filter.

Identity-based substitution parameters map a user's authenticated ID or group memberships to values in a specified column in your data. Values are dynamically substituted into the filter at run time, as appropriate for each requesting user. Here are the supported substitution parameters:

<table>
<thead>
<tr>
<th>Substitution Parameter</th>
<th>Description and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB::SAS.Userid</td>
<td>Determines whether a data value is the same as the requesting user's authenticated ID. empID='SUB::SAS.Userid'</td>
</tr>
<tr>
<td>Note: If the casing of your user IDs might not match the casing of corresponding values in your data, use the upcase function on both sides of the expression. For example: upcase(empID)=upcase('SUB::SAS.Userid')</td>
<td></td>
</tr>
</tbody>
</table>
### Substitution Parameter

<table>
<thead>
<tr>
<th>Substitution Parameter</th>
<th>Description and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB::SAS.IdentityGroups</td>
<td>Determines whether a data value matches any of the requesting user’s group memberships.</td>
</tr>
<tr>
<td></td>
<td><code>FacilityRegion In ('SUB::SAS.IdentityGroups')</code></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The comparison is against each group’s unique name, so your data must</td>
</tr>
<tr>
<td></td>
<td>contain unique group names. In a group definition in SAS Environment Management, the ID</td>
</tr>
<tr>
<td></td>
<td>field contains the group’s unique name.</td>
</tr>
</tbody>
</table>

Here are additional details:

- Only the substitution parameters in the preceding table are supported. (Other substitution parameters that are supported in SAS 9 are not applicable in SAS Viya.)
- Both substitution parameters return values in the casing in which they are received. (In SAS 9, the SAS.UserId parameter always returns values in the uppercase format.)
- You can use the `accessControl.isDebugEnabled` action to see how an identity-driven filter resolves for you.

### Example: User ID Substitution

If a tableB has an empID column with values that match the user IDs with which users authenticate, you might assign this filter to Authenticated Users:

```
empID='SUB::SAS.Userid'
```

At request time, each user’s ID is substituted into the right side of the expression. In a request from userA, the expression resolves as:

```
empID='userA'
```

As a result, userA gets only those rows where the value in the empID column is `userA`.

### Example: Membership Substitution

If tableC has a FacilityRegion column with values that match the unique names for user groups, you might assign this filter to an AllRegions group:

```
FacilityRegion In ('SUB::SAS.IdentityGroups')
```

At request time, each affected user’s list of group memberships is substituted into the right side of the expression. In a request from user13 (who is a member of the grp7, grp9, and AllRegions groups), the expression resolves as:

```
FacilityRegion In ('grp7','grp9','AllRegions')
```

As a result, user13 gets only those rows where the value in the FacilityRegion column is `grp7`, `grp9`, or `AllRegions`.

**Note:** Authenticated Users is not one of the listed memberships, because it is an access control principal, not a user group.

**Note:** The list of memberships is determined when the requesting user’s CAS session starts.
Using Functions in Row-Level Filters

Not all functions work in row-level filters. The following functions are supported:

- upcase
- lowcase
- substr

Other functions might work, but are outside the current scope of support.

Precedence and Cumulative Access

Here are examples of how precedence works for row-level grants:

- If userA has a direct full grant and a row-level grant from groupA, userA can see all rows. The full grant to the user has higher precedence than the row-level grant to the group.
- If userA has a direct row-level grant and a full grant from groupA, userA can see only those rows that are within the direct row-level grant. The row-level grant to the user has higher precedence than the full grant to the group.
- If userA has a full grant from groupA and a row-level grant from groupB, userA can see all rows. The grants have equal precedence, so access is cumulative. In this example, the row-level grant provides only redundant access. Notice that the row-level grant does not limit or constrain the full grant.
- If userA has row-level grants from groupA and groupB, userA can see any row that is permitted by either filter. The row-level grants are at the same level of identity precedence, and userA had no direct setting, so access is cumulative.
- If userA has a full grant from groupA and a row-level grant from Authenticated Users, userA can see all rows. The full grant to the group has higher precedence than the row-level grant to Authenticated Users.

The following figure depicts the authorization decision process for a request by userA to access a CAS table. The figure applies to the Select permission and assumes that userA has ReadInfo access.
Multiple row-level grants provide cumulative access only if all of the following circumstances exist:

- The requesting user does not have a direct access control for the Select permission.
- None of the requesting user’s groups have a direct grant or denial for the Select permission.
- Two or more of the requesting user’s groups have row-level grants.

Note: All custom and LDAP groups have equal precedence, regardless of any nested memberships.

Note: A filter for a row-level grant that is assigned to Authenticated Users is never cumulative (joined with other filters by OR). Authenticated Users is a construct that has lower precedence than any group.

---

**Column-Level Access**

CAS supports column-level permissions, where a user can access some (but not all) columns in a table. You can use the Access Control action set to set column-level permissions. See also Application and Persistence.

To prevent a user from accessing a column, deny the user both the ReadInfo permission and the Select permission for that column. Denying both permissions ensures that the user cannot access the column through any CAS action or interface.
Note: Do not rely exclusively on a denial of the ReadInfo permission on a column to hide that column. Not all CAS actions require the ReadInfo permission at the column level.

**CAUTION**

Not all interfaces can successfully interact with tables that have column-level permissions. Before you provide a production implementation of column-level permissions, verify that results in all applicable interfaces are acceptable.

For example, in SAS Visual Analytics, column-level access is not supported and can yield unexpected results. If userA lacks access to any column that is included in a SAS Visual Analytics report object, userA cannot see any data in that report object.

**Access to Actions**

Action sets and actions that have no access controls are available to all authenticated users. As a result, almost all action sets and actions are available to all users. In general, the ability to perform a particular task is managed by access controls on the target data, not by access controls on actions.

An exception is actions for adding nodes and stopping the server. The initial configuration denies Authenticated Users the Execute permission for those actions. Initially, only Superusers can add nodes or stop the server.

Here are additional details:

- The ReadInfo and Execute permissions are enforced for actions.
- The ReadInfo and Load permissions are enforced for action sets.
- Unregistered action sets are subject to access constraints that are defined on the _UNREGISTERED action set. The initial configuration denies Authenticated Users the Load permission on the _UNREGISTERED action set.

**Note:** An unregistered action set is an action set that is not listed in the database of action sets that SAS provides. SAS solutions use only registered action sets.

**Note:** An attempt to load an action set that does not exist generates an access denied error message, because no such action set is known and registered. For example, if you do not correctly specify the action set name in a load request, an access denied error message is generated.

- The identity type Guest has the same access to non-administrative actions as Authenticated Users. That access is relevant only for sites that choose to enable guest access.
Authorization Decisions

Precedence

In the CAS authorization system, precedence is determined by where an access control is set and who an access control is assigned to.

- Direct access controls have precedence over inherited settings.
- The principal precedence hierarchy is relatively flat. It consists of only the following three levels: 1) individual users, 2) user groups, and 3) the construct Authenticated Users.

Note: All user group memberships are at the same level of precedence, even if groups are nested.

Direct access controls have precedence over inherited access controls, regardless of who the principal is. For example, if only the following access controls exist, then UserA cannot access TableA:

- UserA has a direct grant of ReadInfo on caslibA.
- Authenticated Users has a direct denial of ReadInfo on TableA, which is in caslibA.

Note: One way to enable UserA to access TableA is to add a direct grant of ReadInfo for UserA on TableA. UserA’s direct grant has precedence over the direct denial for Authenticated Users.

How Access Is Evaluated

Each access request initiates an authorization decision process. That process terminates when an outcome is reached. For example, here is the authorization decision process for the Select permission in a request to access data in a CAS table:

1. If there are relevant direct access controls on the table, those access controls determine the outcome as follows:
   a. If there is a setting that is specifically assigned to the requesting user, that setting determines the outcome.
   b. If there is a denial from a group, the outcome is Not Authorized.
   c. If there is a grant from a group, the outcome is Authorized.
   d. If there is exactly one row-level grant from a group, the outcome is Row-Level Authorization (authorized for rows within the applicable filter).
   e. If there are two or more row-level grants from groups, the outcome is Row-Level Authorization (authorized for any row that is within any of the applicable filters). See Multiple Filters and Cumulative Access.
   f. If there is a setting for Authenticated Users, that setting determines the outcome.

2. If there are no relevant direct access controls on the table, direct access controls on the parent caslib determine the outcome as follows:
If there is a setting that is specifically assigned to the requesting user, that setting determines the outcome.

b If there is a denial from a group, the outcome is Not Authorized.
c If there is a grant from a group, the outcome is Authorized.
d If there is a setting for Authenticated Users, that setting determines the outcome.

3 If there are no relevant direct access controls on the table or the parent caslib, the outcome is Not Authorized.

Access Control Transactions

If you want to preview the results of changes to CAS access controls before you save those changes, use an access control transaction. Here are examples:

- When you have unsaved changes in a CAS object’s Authorization window, you can click Preview. Review the results, and then save or cancel your changes.
- When you manage CAS authorization using the command-line interface, you can choose to check out an object, modify its access controls, and then commit or roll back the changes. In the command-line interface, a transaction is automatically started when you check out an object.
- When you manage CAS authorization programmatically, you can choose to start a transaction and check out one or more objects before you make changes. When the transaction is open, any whatIsEffective actions that you run incorporate the effects of your unsaved changes. Review the results, and then commit or roll back your changes.

Here are key points about access control transactions:

- This feature is for only changes to access controls. This feature does not provide transaction support for interactions with data or with any other aspect of CAS objects.
- In SAS Environment Manager, access control transactions are used when CAS access controls are managed.
- In programmatic and command-line interfaces, use of access control transactions is optional. You do not have to use a transaction in order to modify CAS access controls. Use access control transactions if you want to preview the results of your changes or ensure that nobody else is modifying access controls for the same objects at the same time.

Origins of Effective Access

Origins information explains effective access by answering the question: Why does this identity have this effective access to this object?

Origins information identifies the highest precedence access control that causes the access outcome for a particular identity, object, and permission. If there are multiple tied highest precedence access controls, origins information includes all of them. Additional, lower precedence controls are not included.

The following table provides simple examples of origins information. Each row in the table is for a different (independent) scenario. In each example, we are looking at why UserA has an effective access value of Not Authorized for the ReadInfo permission on TableA. UserA is a member of GroupA and GroupB. TableA is in CaslibA.
### Origins: Examples

<table>
<thead>
<tr>
<th>Highest-Precedence Access Control (or Controls)</th>
<th>Origins Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>On TableA, a direct denial for UserA.</td>
<td>TableA UserA</td>
</tr>
<tr>
<td>On TableA, a direct denial for GroupA.</td>
<td>TableA GroupA</td>
</tr>
<tr>
<td>On TableA, direct denials for GroupA and GroupB.</td>
<td>TableA GroupA, GroupB</td>
</tr>
<tr>
<td>On TableA, a direct denial for Authenticated Users.</td>
<td>TableA Authenticated Users</td>
</tr>
<tr>
<td>On CaslibA, a direct denial for UserA.</td>
<td>CaslibA UserA</td>
</tr>
<tr>
<td>On CaslibA, a direct denial for GroupA.</td>
<td>CaslibA GroupA</td>
</tr>
<tr>
<td>On CaslibA, direct denials for GroupA and GroupB.</td>
<td>CaslibA GroupA, GroupB</td>
</tr>
<tr>
<td>On CaslibA, a direct denial for Authenticated Users.</td>
<td>CaslibA Authenticated Users</td>
</tr>
<tr>
<td>There are no relevant access controls.</td>
<td>Caslib default</td>
</tr>
</tbody>
</table>

**TIP** To obtain origins information in the Authorization window, see *Identify the Source of Effective Access*.

## Reduced Visibility: Hidden Caslibs

A hidden caslib is omitted from most lists of caslibs. Tables in a hidden caslib are omitted from most lists of tables. A caslib is hidden if its `hidden` parameter is set to `true`. You can set the `hidden` parameter in the `tables.addCaslib` action.

Hiding a caslib does not protect the caslib or limit access to the caslib’s data. Hiding a caslib just reduces visibility by preventing the caslib and its tables from being listed in certain contexts. In those contexts, hidden caslibs and their tables are unlisted for all users and administrators, regardless of roles and access controls. Hiding a caslib affects everyone equally.

For example, two of the predefined caslibs (AppData and ReferenceData) are hidden. Users can use the data in those caslibs because appropriate predefined access controls are in place. However, those caslibs (and their tables) are not listed in most contexts. They are excluded from selection lists in SAS Visual Analytics.

**TIP** Hide a caslib only if you have users who must be able to use that caslib’s data but should not see that caslib or its tables in most lists.

Here are examples of contexts in which hidden caslibs are listed:
On the **Data** page in SAS Environment Manager, hidden caslibs are listed.

In the `tables.caslibinfo` action, hidden caslibs are listed if you specify the value `true` for the `showHidden` parameter.

---

**Application and Persistence of CAS Access Controls**

**About This Topic**

This topic applies to direct access controls at the table, column, and row levels. For conciseness, this topic refers to all such controls as table-level controls.

This topic refers to table-level controls being *set on or persisted to* a source file or in-memory table. That phrasing is an abstraction. On disk, CAS access controls are written to an item store. In the item store, each table-level control is associated with a metadata representation of either a source file or an in-memory table. For more information about the item store, see “cas.PERMSTORE=’path’” in *SAS Viya Administration: SAS Cloud Analytic Services*.

**Name-Based Controls**

**Introduction**

In CAS authorization, each table-level control is associated with either a source file or an in-memory table.

In most cases, table-level controls are associated with source files. For simplicity and predictability, the best practice is to set controls on source files, because that ensures that the controls are associated with source files. Such controls are called source-based controls.

In certain specialized workflows, table-level controls are instead associated with in-memory tables. Such controls reference in-memory tables by their names, so they are called name-based controls. The following sections explain how name-based controls are created and used.

**Creation of Name-Based Controls**

In general, each table-level control that you set is tied to a source file for one of the following reasons:

- When you set the control, you specify or select a source file (for example, cars.sashdat).
- When you set the control, you specify or select an in-memory table (for example, CARS), and CAS persists the control to the corresponding source file (for example, cars.sashdat).

The exception, in which table-level controls are name-based, occurs in the following circumstances:

- You set controls on an in-memory table that has no corresponding source file.
- You set controls on an in-memory table that is disconnected from its source file (*cross-caslib data*).
Table 5  Summary: Creation of Name-Based Controls

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Is a Name-Based Control Created?</th>
</tr>
</thead>
<tbody>
<tr>
<td>You set a direct access control on a source file.</td>
<td>No</td>
</tr>
<tr>
<td>You set a direct access control on an in-memory table that was loaded from a source file in the same caslib.</td>
<td>No</td>
</tr>
<tr>
<td>You set a direct access control on an in-memory table that was loaded from a source file in a different caslib.</td>
<td>Yes</td>
</tr>
<tr>
<td>You set a direct access control on an in-memory table that has no corresponding source file.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**IMPORTANT**  The best practice is to set table-level controls on persisted data (data that has been saved as a source file within a caslib), not on in-memory data.

Usage of Name-Based Controls: Data Access

In general, table-level protections for in-memory data are provided by source-based controls. The exception, in which table-level protections are provided by name-based controls, occurs in the following circumstances:

- The in-memory table has no source file.
- The in-memory table has a source file that has no direct access controls.

**Note:** Name-based controls are used in this circumstance only after an August 2020 hot fix.

- The in-memory table is disconnected from its source file. (The source file is in a different caslib.)

Table 6  Summary: Usage of Name-Based Controls for Access to In-Memory Data

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Are Name-Based Controls Used?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The in-memory table has a connected source file that has direct access controls.</td>
<td>No</td>
</tr>
<tr>
<td>The in-memory table has a connected source file that has no direct access controls.</td>
<td>Yes¹</td>
</tr>
<tr>
<td>The in-memory table has no connected source file.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

¹ After an August 2020 hot fix.
Usage of Name-Based Controls: Table Loads

Note: This section applies after an August 2020 hot fix. Prior to that hot fix, name-based controls are never used in evaluation of requests to load tables.

In evaluation of a request to load a table, name-based controls are used only if all of the following circumstances exist:

- The source file has no direct access controls.
- The destination in-memory table has name-based controls.
- The destination in-memory table is in the same global caslib as the source file.
- The load table request meets all of the following criteria:
  - The request specifies `promote=True`.
  - The request does not specify the `vars` parameter.
  - The request does not specify the `where` parameter.
  - The request either does not specify a name for the destination table or specifies the default name for the destination table (in the `casOut` parameter).

Note: The default name is source file name in all upper case characters, with any recognized file extension removed and any slashes (/) replaced with periods (.).

Disadvantages of Name-Based Controls

Although name-based controls can be useful in certain specialized workflows, relying on name-based controls is not a preferred approach. Here are key points to consider before using name-based controls:

- In the context of data access, name-based controls can be circumvented by specifying a different destination name in a load action.
- Name-based controls can duplicate or conflict with source-based controls, which might cause confusion or yield unintended results.

TIP

If any source-based control exists for a table, no name-based controls are examined for that table. This differs from access control precedence, which examines only those controls that are relevant to a specific request.

- Name-based controls are created in only specific circumstances, and it is not always obvious whether those circumstances exist.

TIP

If you want to avoid creating name-based controls, use the following practices:

- When you set access controls using a CAS action or command-line interface, specify a source file, including its file extension, in the table parameter.
- When you set access controls using SAS Data Explorer, open the CAS Authorization window from a source file, not an in-memory table.
Name-based controls are used in only specific circumstances, and it is not always obvious whether those circumstances exist.

**TIP** The best way to avoid use of name-based controls is to make sure they are never created. If you want to minimize use of any name-based controls that are created, use the following practices:

- Avoid workflows that provide access to cross-caslib data.
- Avoid workflows that provide access to unpersisted in-memory data.

Name-based controls are not included in responses to requests for authorization information for a source file. For example, any name-based controls for the in-memory table CARS are not included in a response to a request to list controls for the source file cars.sashdat. Similarly, any name-based controls for the in-memory table CARS are not included in a response to a request to describe effective access to the source file cars.sashdat.

**Persistence of Access Controls**

**When You Save Data**

When you save data, any direct access controls are managed as follows:

- If the original source file is replaced, direct access controls are persisted. A replacement can occur only for a source file that matches the specified output name and extension. Here are details:
  - For the file extension, replacement uses SASHDAT by default. For example, if the specified output name is cars, the resulting source file (cars.sashdat) can replace only an existing source file cars.sashdat, not an existing source file cars.csv.
  - For the file name, on a case-sensitive file system, replacement requires a case-sensitive match. For example, if the specified output name is cArS, the resulting source file on Linux (cArS.sashdat) can replace only an existing source file cArS.sashdat, not an existing source file cars.sashdat.

  **Note:** If you do not meet the case-sensitive match requirement, you might have multiple source files that differ only in the casing of their names. In that situation, it is useful to understand that source file names are not case sensitive in the context of authorization. For example, if a caslib has loaded tables from two source files (cars.sashdat and cArS.sashdat), authorization processes do not distinguish one source file from the other. A single set of access controls applies to both source files and their loaded tables. Adding or deleting access controls for any of those entities affects all of those entities.

- If a new source file is created, direct access controls are not persisted to the new source file. Access to the new source file is determined by access controls on its parent caslib.

- If a different existing source file is overwritten, direct access controls are not persisted. Access to the existing source file is unaffected.

Here is an example:

1. You add access controls to the source file CARS.csv.
2. You load the source file CARS.csv to global scope.
3 You modify the in-memory table CARS (for example, you add a calculated column).

4 You save the in-memory table CARS, specifying to replace any existing same-named source file. In the save request, you do not specify a file extension.

5 A new source file CARS.sashdat is created. The access controls that you set on the source file CARS.csv are not replicated on the new source file.

Note: In this example, a replacement did not occur because the in-memory table was loaded from a CSV source file and you did not specify the CSV file extension as part of the output name in the save request.

In workflows that involve saving data, the best practice is to establish and maintain any direct access controls on the target (post-preparation) files. By default, such access controls survive activities such as deletion and replacement.

The Ambiguity Problem

If you specify or select an in-memory table when you set access controls, the results might be unexpected. In a caslib that contains multiple same-named source files, it might not be obvious which of those source files corresponds to the table that is currently in memory.

Here is an example:

1 caslibA includes the source files CARS.csv and CARS.sashdat.

2 Someone loads CARS.csv to global scope.

3 In SAS Data Explorer, someone else specifies or selects the in-memory table CARS when setting access controls, incorrectly assuming that they are protecting the source file CARS.sashdat.

4 The new access controls are written to the source file CARS.csv, not to the source file CARS.sashdat.

Considerations for Rows and Columns

Row-level filters and column-level access controls are applied to requests to access or save data, not to requests to load data. (There is an exception for loading of cross-caslib data.)

For example, if tableA has a row-level filter that enables userA to see only those rows where the value in the MAKE column is Ford, the filter is applied as follows:

- If userA loads tableA, all rows are loaded.
- If userA accesses tableA, userA sees only those rows where the value for MAKE is Ford.

As explained in the preceding sections, row-level filters and column-level access controls are not persisted when a user saves a table as a new or different table. The following information continues the preceding example:

- If userA saves tableA to the file from which tableA was loaded, the replaced table contains only those rows where the value for MAKE is Ford. This reduction in scope affects all subsequent loads of tableA and all access by subsequent users. Some users might access fewer rows than intended. Because the row-level filter still exists on the source file and can further reduce access, no users access more rows than intended.
- If userA saves tableA to a different file, the new source file contains only those rows where the value for MAKE is Ford. This reduction in scope affects all subsequent loads of tableA and all access by subsequent users. Some users might access fewer rows than intended. Because the
Considerations for Cross-Caslib Data

In most workflows, data does not move across caslibs. For example, when you load and promote data, you are usually performing a same-caslib activity, moving data from session scope to global scope in the same caslib.

In a few workflows, data does move across caslibs. For example, if you promote data from one caslib to another, or copy data from one caslib to another, you are performing a cross-caslib activity. The target data that results from a cross-caslib activity is referred to as cross-caslib data.

Cross-caslib data has no ongoing relationship to the source (original) caslib and file. Changes to access controls on the source caslib and file have no effect on cross-caslib data.

Access to cross-caslib data is subject to access controls on the target caslib and file. Avoid performing cross-caslib activities on sensitive data. If you must perform cross-caslib activities on sensitive data, make sure the target caslib and file have appropriate protections.

TIP  If the source data for a cross-caslib activity has fine-grained constraints (row-level filters or column-level denials), make sure the cross-caslib activity is performed by an identity that has sufficient access to the source data. Any constraints on the source data are applied during the cross-caslib activity, and are based on the identity that performs the cross-caslib activity. Subsequent access to the cross-loaded data cannot be expanded to include data that was unavailable to the identity that performed the cross-caslib activity.

CAS Authorization: Host Access

Why Host Access Matters

If host-layer access requirements are not met, grants in the CAS authorization layer do not provide access.

If host-layer access protections are not in place, denials in the CAS authorization layer do not fully prevent access.

Which Host Account Matters

The account under which a CAS server process runs must have appropriate host-layer access to target directories and files. Depending on context, that account is either an individual user’s personal host account or the CAS server’s shared service account. The following table provides examples:
### Context

<table>
<thead>
<tr>
<th>Identity of the CAS Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>A request from the SAS Visual Analytics web application to a CAS server on Linux.</td>
</tr>
<tr>
<td>A request from the SAS Visual Analytics web application to a CAS server on Windows.</td>
</tr>
<tr>
<td>A request in a programming-only deployment.</td>
</tr>
</tbody>
</table>

¹ Unless the requesting user is a member of the custom group CASHostAccountRequired.

### Host Access in a Programming-Only Deployment

In a programming-only deployment, every user accesses a CAS server’s host files and directories using their individual host account. Users must have host access, so it is possible for them to access the back-end machine directly, bypassing the CAS authorization layer.

If you have sensitive data, ensure that all CAS access distinctions are mirrored in the host authorization layer. For example, if you use the CAS authorization system to deny userA Read access to a path-based caslib called caslibA, you must also set up host access controls that prevent userA from accessing that caslib (directory). Without such protection, userA could use a host command to copy files from the caslibA directory to the directory for a caslib that userA can access from CAS.

### Host Access in a Full Deployment

**Note:** This section applies to a CAS server on Linux. Host access from a CAS server on Windows is always under each user’s individual identity.

In a full deployment, host access from CAS is sometimes under individual identity and sometimes under a shared identity. Here are details:

- For users who access CAS only from a programming interface such as SAS Studio, all host access from CAS is under each user’s individual identity. For such users, you must mirror CAS layer access distinctions in the host layer.

- For users who access CAS only from a visual interface such as SAS Visual Analytics or SAS Environment Manager, all host access from CAS is under a shared identity. Such users need CAS layer access to data, but they do not need host access to data. Only the shared identity needs host access to data.

  For such users, there is no reason to create host access controls that mirror your CAS access controls. Of course, you should always host protect your resources in accordance with your security requirements.

- For users who access CAS from both types of interface, host access and experience vary depending on the type of interface that is used. For example, the personal caslib that they use from programming interfaces is not automatically accessible from the visual interfaces.
You can align access and experience for such users by assigning them to the CASHostAccountRequired group. For members of that group, host access from CAS is always under individual identity. Before you use this approach, review the associated limitations. See SAS Viya Administration: Identity Management.

Using CAS to Modify Host Access

You can use CAS to add host access controls to a new directory or file in the following circumstances:

- You add a caslib of the type PATH or DNFS and specify to create a directory (on Linux).
- You save a table to a file (on Linux).

This functionality is provided by the permission parameter in the CAS actions tables.addCaslib and tables.save.

Here are the available fixed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Octal¹</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>700</td>
<td>Grants Read and Write access to only the owner.</td>
</tr>
<tr>
<td>GroupRead</td>
<td>750</td>
<td>Grants Read and Write access to the owner.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grants Read access to the owning group.</td>
</tr>
<tr>
<td>GroupWrite</td>
<td>770</td>
<td>Grants Read and Write access to the owner and the owning group.</td>
</tr>
<tr>
<td>GroupWritePublicRead</td>
<td>775</td>
<td>Grants Read and Write access to the owner and the owning group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grants Read access to everyone.</td>
</tr>
<tr>
<td>PublicRead</td>
<td>755</td>
<td>Grants Read and Write access to the owner.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Grants Read access to everyone.</td>
</tr>
<tr>
<td>PublicWrite</td>
<td>777</td>
<td>Grants Read and Write access to everyone.</td>
</tr>
</tbody>
</table>

¹ These octal values are for directories. For saved files (SASHDAT and CSV), the executable bit is not set.

Here are additional details:

- The owner is the host account that creates the directory or file. The owner always gets full access, regardless of whether you use the permission parameter. Use the permission parameter to further refine access.
- The owning group is the host group that is the primary group for the owner.
- You can specify an octal. You are not limited to the fixed values that are listed in the preceding table.
Access to Files in the Public Caslib

Initially, all users have host-layer Write access to the directory for the Public caslib. To limit host-layer Write access to that directory, adjust host-layer access controls.

For example, on a Linux directory you can set a user ownership access flag that is called the sticky bit. If the sticky bit is set for the Public caslib’s directory, only the host account that creates a file in that directory can remove that file. If the sticky bit is not set for the Public caslib’s directory, any user with Write access to the directory can remove files from that directory.

CAS Authorization: Guidelines

The following guidelines can contribute to simplicity and security:

- Minimize use of individual tables as targets.
- Minimize use of individual users as principals.
- Perform a backup before and after you make significant changes to your system.
- Do not set unnecessary denials. Any access that is not granted is implicitly denied.
- If you deny someone access to part of a table (using a column-level or row-level access control), make sure that identity cannot update or insert rows in that table.

CAS Authorization: Troubleshooting

Unrecognized Principals

If the Authorization window displays a warning icon next to a principal’s name, that principal does not exist in the identities service.

- If the principal is a host account (for example, cas) that does not exist in your LDAP provider, you can ignore the warning icon.
- If the principal is an internal service account (for example, sas.ops-agentsrv, sas.searchIndex or sas.search), you can ignore the warning icon.
- If the principal should still exist in the identities service, make sure the identities service can still contact your LDAP provider.
- Otherwise, consider deleting the access controls that are assigned to the principal that no longer exists.
Note: Deletion of a custom group does not cause automatic deletion of rules in which that custom group is the principal.

Inability to Modify Access

- Make sure the target object is global-scope (not in a personal caslib or session scope).
- Make sure you have the ManageAccess permission for the target object. Or, assume a role that is exempt from that permission requirement. See SAS Viya Administration: SAS Cloud Analytic Services.
- If you get the following error: "The access controls for table {name} are being blocked by a loaded table of the same name", select or specify a source file (for example, tableA.sashdat), not an in-memory table. See "Application and Persistence of CAS Access Controls".
- If you get the following error: "The object {name} {type} is currently locked, so you cannot modify its access controls", the target object is already participating in an active access control transaction. Either wait for the lock to be released, or terminate the CAS session in which the object is locked. See “Access Control Transactions”.

Unintended Loss of Access

Reinstate Access: Instructions for Users

If you inadvertently block your own access to an object, contact an administrator for assistance.

Note: Anyone who has the ReadInfo and ManageAccess permissions for the object can reinstate your access.

Reinstate Access: Instructions for Administrators

1. In the applications menu (≡), under ADMINISTRATION, select Manage Environment.

   Note: These instructions use SAS Environment Manager. You can instead use an alternate interface.

2. In the navigation bar, click 🌐.

3. On the Servers page, right-click on a CAS server, and select Assume the Superuser role.

   Note: If the Assume the Superuser role action is not available, you are not a member of the Superuser role for the selected server. If your Superuser role membership is exclusively through the SAS Administrators group, make sure you have opted in to that membership in your current session.
CAS Authorization: Interfaces

All CAS authorization requirements and constraints are always fully enforced. However, not all interfaces expose all CAS authorization features.

In the following table, the shaded part of each circle is an approximation of the amount of CAS authorization functionality that a particular interface exposes. The shading indicates relative coverage. The shading does not indicate alignment of coverage across interfaces.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Control action set</td>
<td>Programmatic interfaces for CASL (the CAS procedure), Python, Lua, and R. See SAS Viya: System Programming Guide.</td>
</tr>
<tr>
<td>REST API</td>
<td>The REST interface for CAS.</td>
</tr>
<tr>
<td>SAS Java Client Interface for SAS Viya</td>
<td>The Java programming interface for CAS actions.</td>
</tr>
<tr>
<td>Command-line interface</td>
<td>A simple, scriptable interface that includes commands for managing access at the caslib, table, and row levels in a full deployment.</td>
</tr>
<tr>
<td>Authorization window</td>
<td>A visual component for managing access at the caslib, table, and row levels.</td>
</tr>
<tr>
<td>CAS Server Monitor</td>
<td>A web application for managing access at the caslib level. See SAS Viya Administration: Using CAS Server Monitor.</td>
</tr>
</tbody>
</table>